The Amazingly Well Written and Informative Chuzzle Postmortem

Part 1. John hits the Keyboard

Okay, here's a brief history of the trials and tribulations of making Chuzzle. There actually weren't many tribulations in making Chuzzle that weren't intentional human error, or people just trying to make eachother's lives miserable, but this is an opportunity for me detail some of the development process for other developers. This won't be very interesting to a non-developer. It may not even be interesting to a developer!

Chuzzle began as a faint idea over Summer 2005, and became a game mockup around Sept 15-20th 2005 (don't remember the exact date). I always have a lot of difficulty making a "game mockup" as opposed to jumping right in with the game's full graphics, but to interest Popcap, I wanted a version of the game mechanic out and ready to go as soon as possible. I sent a version of Chuzzle with a 10x10 grid made of sliding blocks. Jason Kapalka at Popcap was interested, so we set about looking for a theme.

My wife suggested sliding fuzzy caterpillars that match up and turn into butterflies. Jason Kapalka suggested a slot machine type theme-- sliding lemons and cherries and suchlike. Neither of these really lite the fire for me, and, complicating things, I was going through some bizarre Pac-man nostalgia phase. For fun, I made a version of the game where you slid Pac-man ghosts--- when you matched them up, they'd turn blue, and a Pac-man would zoom in and eat them, leaving the eyes to fly offscreen. For a while, I was so enamored with this idea that I wanted to try to reach Namco and beg for the rights. My wife came in, made a face, and said, "That's no fun, you should use fuzzy caterpillars."

The original Chuzzles were rendered without any special plugins, and they looked like this:

It's just a sphere in 3DS Max, messed up with a noise modifier.

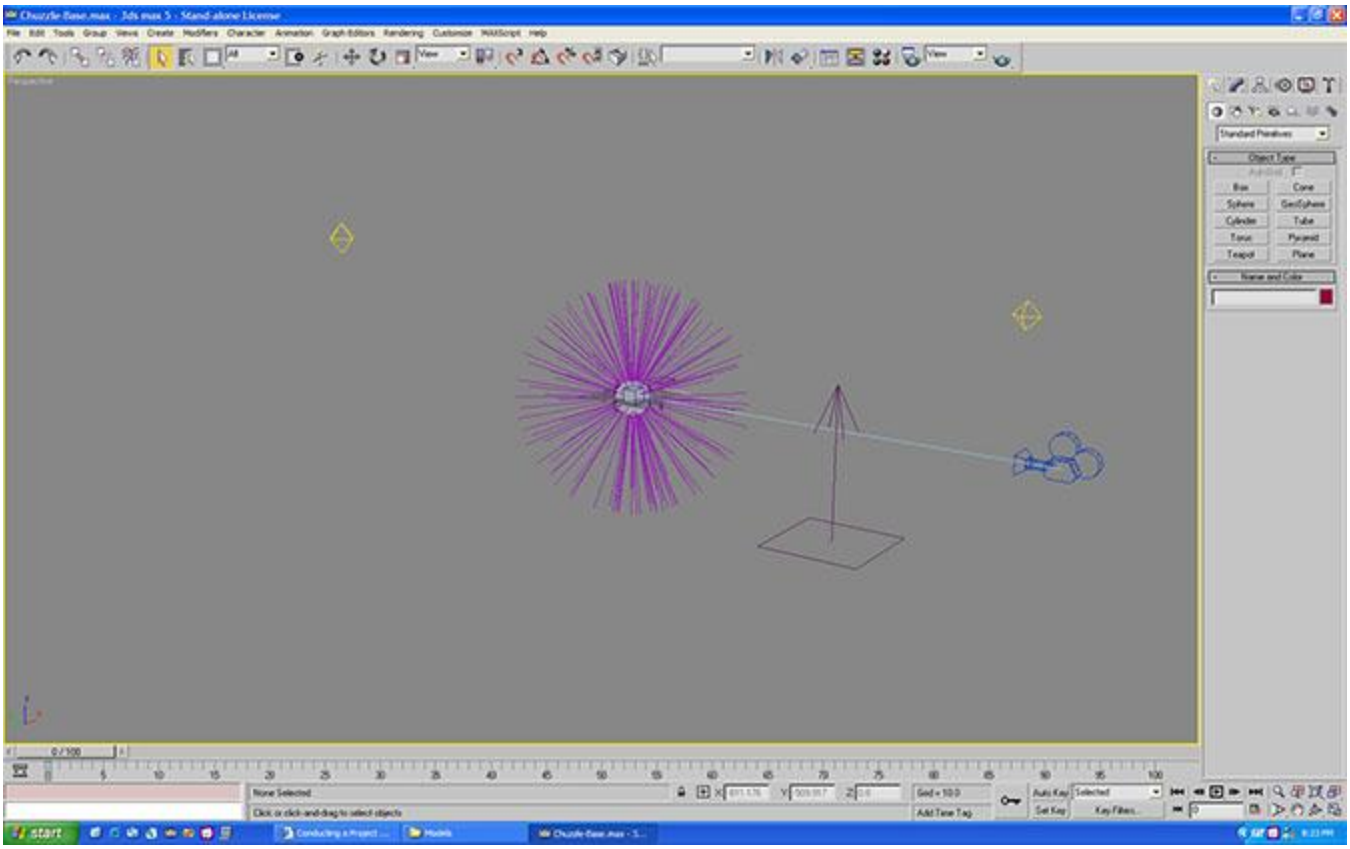
I started working a bit with my artist friend Gary Clair on making the Chuzzles more interesting. Gary is a concept artist of astounding talent, and *his* wife suggested that the game be given a 70's theme. Soon, we were looking at this mockup:

It was also around this time that my male friends started goofing on me about the name "Chuzzle."

Anticipating issues with colorblindness, and going a little bit over the edge with the irrational enthusiasm, I played with the Chuzzles wearing costumes to help identify color. So, the very first playable Chuzzle with fuzzbals looked like this:

This wasn't even playable for ten minutes. It became very difficult to find matches-- the costumes were too distracting. And, putting the costumes on the Chuzzles by color gave the game a horrid, offensively symmetrical aesthetic.

Around this time, I got the Shag Hair plugin and made a fuzzy ball using it. What I immediately saw what that the Chuzzle's hair could pulse and ripple and otherwise react when you slid them around. Soon I had Chuzzles rendered in 8 colors with about 200 animation frames each, taking up approximately 10mb of disk space *(ed. note: larger than a full game would be in those days).* But do first, optimize later! This is how a Chuzzle looks in 3DSMax, btw. The purple thing is the Chuzzle itself (as you can see, the core is very small). The rectangle with the arrow sticking up out of it is a wind modifier-- when you're seeing a Chuzzle poof from getting dragged in the game, you're actually watching "wind" blow at it from the other direction.



There was a problem now. With the Chuzzles pulsing and moving, the costumes looked horribly static, like cardboard overlays stuck over a movie. There was no real way around this. The costumes were tossed.

Now that all the graphic concepts were done, I cranked up some old music and started to code and make up menus and sound effects. People have asked about the Chuzzle sound effects, which sound very retro-arcadey. Here's how they were made: I downloaded an Atari 800 emulator (one of the machines of my strapping youth), and started writing little Basic programs to make sound, like:

```
5 VOLUME=10
10 FOR X=0 TO 100
20 SOUND 0,X,10,VOLUME
30 SOUND 1,X,8,VOLUME
31 VOLUME=VOLUME-0.1
40 NEXT N
```

And boy, let me tell you, in this day of text editors and fast code, you'd be blown away thinking about what we had to work with back in those days. Anyway, using Atari Basic, I managed to make some almost pleasing sound. But, using Goldwave to give them slight speed modifications and flanges and various mixing made them exactly what I was looking for. So don't bother trying to duplicate a Chuzzle sound in Atari basic-- every sound is actually 2-3 basic program sounds mixed together (some very similar) and peppers with so many effects and adjustments that even I

couldn't duplicate them now. But I did capture that "retro arcade bleeps and bloops" I was looking for.

Okay, so, in about a month and I half, I had a "finished" game ready for Popcap (by "finished" I mean, I could have released it on my own site and started selling it. In the context of Popcap, "finished" here means "ready to start on it.")

Here's what Chuzzle ver 1.0 Raptisoft looked like-- these are all my 1.5 month graphics with no Popcap makeover (click to enlarge).

So you can see, it looks different, but the same. Sort of embryonic.

And yeah, some of you will have noticed "Chuzzle Duel" there on the menu. That last screenshot is from "Chuzzle Duel," which was a completely multiplayer version of Chuzzle, thrown in, if you believe them, just to give Popcap ulcers. It was eventually taken out because it was considered to be a tech support and QA nightmare, alas. (I am currently petitioning them to allow me to release it as freeware). Basically, what you did was make matches, while your opponent did the same (you could watch him in that little window over the bottle). Whenever you filled the bottle completely, you were given a "bomb" that you could drop on your opponent (a bomb consisted of two locks getting thrown onto his screen). Accumulating bombs so that you could drop 6-8 locks on him at one time was the order of the day-- your choice was "use locks now and prevent him from making as many matches" or "store up locks in the hopes of hitting him with a no-more-moves." Nobody ever "lost" in this-- you basically played until you were finished, checking who had the highest score, or most locked boards or whatever.

Although my wife and I enjoyed it, in the end I caved to Popcap's wishes and put Mindbender in instead.

Now, some stats for the developer-minded stats freak! Chuzzle had this time looked like this:

Required DirectX8.1
Did not support software only mode
Was written in a modified version of the Hamsterball framework, re-tooled to resemble the Popcap framework
Was about a 15mb download (all graphics were PNGs at this point)
Ran at about 60-65 FPS
Could handle multiple resolutions without code changes (via stretching)
Had multiplayer (both IPX and TCP-IP)

So like I said, it was "complete." After optimizing the graphics for size, I figured it could go out. So optimistic was I that the journey was nearly done that I went and optimized the graphics myself. I got Chuzzle down to a 2mb download, and even managed to make an ActiveX version. Surely it was now time to rest.

Part 2. Popcap Squashes me like a bug

"DirectX8 is not acceptable," says Jason Kapalka to me. That's how he talks, really. Bold, bald statements. An iron wall of will, as in you will be defeated. "Why don't you move it to the Popcap framework?"

Hm, why didn't I? The Popcap framework is a very good coding framework, but when I wrote Chuzzle I starting doing some fancy-schmancy things with special effects that weren't common in 2d games back then. For instance, a lot of my effects in Chuzzle involve complicated matrixes to rotate, scale, orbit, and otherwise twist graphics into all kinds of shapes. I'd worked with the Popcap framework extensively, since Dynomite, but I had coded the Chuzzle framework to essentially be the Popcap framework-- conformed to my extreme laziness and ennui. So, for

instance, in the Popcap framework, you make a button, and then maintain its state-- setting up a Widget to move the button, handle the button, destroy the button, etc. In my own framework, you make a button with 1-2 lines and forget it ever happened. I had implemented a whole lot of things that were completely geared toward an "capricious artist programmer" whose creative bursts were often short circuited by having to write 50 lines of code across three classes just to make a piece of the game UI work.

Plus I was doing all those matrix math transformation for the special effects. The Popcap framework (at that time) couldn't do that. And these effects are very subtle effects that I felt were necessary to keep the whole thing going. Did you know all the Chuzzles are drawn as parallelograms that change their corners randomly by about .5 pixels? Probably not, but in game it makes it look a lot more organic than if they were just static. You can see the difference by playing it in "software mode" vs. "hardware mode." There's no effect in software mode-- see the difference?

Other factors existed: in my previous machinations with Popcap, the memory footprint of the game became an issue (the Popcap framework has a double memory footprint for graphics-- the framework keeps a version around, and managed DirectX keeps a version around). In my framework, only the managed version was around, so I was free to make more graphics without having to worry about overstepping a 64mb limit while in software mode. At this time, the Chuzzles were not colorized-- they were all pre-rendered, every color, because I was able to make them look a little finer that way.

And besides, I loved my multi-resolution option.

From my perspective, moving it to the Popcap framework would be impossible without cutting some of the quality out of the game.

"I'll move it to DirectX7," I said, "but I don't think I can make it work with the Popcap framework as it."

This didn't go over great. I don't remember all the details of this particular battle. I don't even remember how I won it, because if I had it to do over again, I would just move it to the Popcap framework and be done with the game three months earlier. So an empty victory; Jason will be sighing with pleasure somewhere as he reads these words. *(John from 2025 note: I most definitely should have moved it to the Popcap framework)*

The move to DirectX7 went fairly smoothly-- *at this time*. Anyone who has worked in DirectX8 and then dropped down to DirectX7 will know what I went through here. You go from a DirectX incarnation that allows you to initialize DirectX with one line to an incarnation that requires a carefully followed 12 step horror story just to get things to the point where you can draw on the screen. So, when I say "fairly smoothly," I mean it took about two weeks, as in "can you believe I dropped it from DX8 to DX7 in only two weeks?!"

There were some side effects. DX8 does an astounding job of managing textures and making sure reinitialization goes okay. In DX7, I found that the easiest way to go from fullscreen to

windowed (and vice versa) was just by reinitializing everything all over again (more on this stuff later).

So, dropping to DX7 wasn't hard. Then Jason drops the other bomb on me:

"Don't you have a software mode," he says?

Well, I didn't. I was of the opinion that software mode was overrated. Chuzzle is so very low-polygon, surely even a Voodoo card could run it? Does anyone not 3D acceleration these days?

Well, it turns out that some people have machines that won't even allow you to initialize Direct3D! All in all, probably 10-30% of people who buy casual games have a computer configuration such that it will not work properly with hardware acceleration. They'll see graphic glitches, they won't be able to load textures, it'll run slower than it would in software, just all kinds of problems. Okay, so I had to have software rendering. That would be easy enough, except that-- argh, I had done all those matrix transformations. Now I needed a software triangle rasterizer.

Paul Nettle to the rescue. Paul Nettle is the guy who did the physics and collision detection for Hamsterball, and he's essentially a low key nicer version of John Carmack (note: I base my Carmack slander on the book "Masters of Doom"). I commissioned him to write a *fast* triangle rasterizer, that could overwrite a 640x480 screen six times and still maintain a 30FPS framerate on an 800mhz P2. And he did it-- with alpha and everything. How amazing is that?

Now that Chuzzle was down to DirectX7 and could render software, it was time to begin some Popcap QA and beta testing. Jason Kapalka has told me that I should throw down some holy hosannas to the Popcap QA department in this Postmortem, and I will do that, difficult though it may be. There's a natural antipathy between developers and QA. We developers feel that QA keeps hitting us with pointless minor bugs, such as the program crashing every startup, and QA feels that we developers keep putting things in programs that make them run less efficiently, like graphics.

That said, the Popcap QA department is like a finely forged sword: they cut right into the heart with no effort whatsoever. They have some magical method to their madness that literally lets them find *everything*. "Hey, if you drag a row of Chuzzles, then right click, then double left click, and, still holding the right mouse button down, it'll crash if you're dragging the Chuzzle in the upper left corner." This was an actual bug. I had trouble duplicating it myself because I just don't have the reflexes. But lo and behold, if you do manage to duplicate this perfectly, it tries to reference a Chuzzle at array position -1,-1, and boom, crash.

Chuzzle had very few major game bugs. What Chuzzle had a lot of were bugs caused by my decision to not use the Popcap framework. 99% of these are bugs that you or I would never notice, because we've bought a new computer since 1995. Popcap has a QA setup with literally every machine possible. And what DirectX does on some of these older machines makes you wonder how anyone started using DirectX at all.

Popcap's framework is essentially a software framework with a hardware layer thrown on top of it by the Popcap employee that everyone calls "Ace" because calling him "The m'fer who can do freaking anything, how the hell did he do that" is too long. So, the Popcap framework is well suited for these older machines: in its native state, it was designed for 1995.

Now my own framework (I'll call it "Rapt" to distinguish it) is a hardware framework with a software layer thrown into it by Paul Nettle. This means that my framework, run on a 1995 machine, is already at a disadvantage. Well, we found a few small bugs related to this, and I fixed them, and I sort of thought the issue was closed. Little did I know that QA had only touched the tip of the iceberg of their testing process. But they went on to other things, and I set happily about finishing up Chuzzle. Ignorance is bliss.

So, we're looking at the time period from about mid-October to mid-December here. I had had hopes of finishing up Chuzzle for Christmas, but, well, you know. There's a reason Hope was at the bottom of Pandora's box, because it's the worst of 'em.

Part 3. Let's make it do *everything!*

Okay, so final note from 2025 John: I ended the original postmortem here because I started a new game and just never ended up returning to it. What I would have written here was a series of (what I considered very unpleasant) iterations where we did things to Chuzzle to make it more "appealing" or "understandable." A list of a few things: Making it so non-matching moves were allowed, making Chuzzles have to be in a straight row/column to match, and my own personal least favorite, REMOVING THE LOCKS FROM THE GAME and rebalancing it completely in some insane race to the bottom of casual video game difficulty.

I can't think of anything from this era that remained in the game, it was an entirely pointless December-thru-March playing around so that everyone could get their two cents in. I was embittered by this because this delay allowed a Chuzzle-like game (called Barnyard Invasion) to beat Chuzzle to market with a similar mechanic (it played quite differently but did involve sliding rows) and I was convinced the game was going to take a hit from that. That ultimately didn't happen, but those months were so full of "okay, change this" followed by "ugh, change it back" that I got exhausted, even panicked. Artists and programmers are not the most stable of people and when you're both you're getting logarithmic.

The constant fiddling of code and huge modification of core game elements ended up introducing bugs that wouldn't have existed otherwise and I ended up being way too sarcastic and mean spirited in my complaining about new bug reports... which was too much for Popcaps' head tester (who is simply the best person I've ever seen at his job.) I apologized him to him after publication, but wanted to include this to emphasize how frazzling the final few months of Chuzzle's dev was.